IN THE CLAIMS

Listing of Claims:

1. A method for efficiently shifting operands for processing, the method comprising:

~~a)~~ receiving an operand;

~~b) determining~~ selecting whether to apply one of a sixteen bit shift value or no shift value ~~to be applied~~ to the operand at a first stage;

~~c)~~ shifting the operand by the selected shift value;

~~d)~~ outputting the shifted operand to a second stage to select whether to apply one of an eight bit shift value or no shift value;

outputting a result from the second stage to further stages in series, wherein a maximum shift value at each successive stage decreases relative to a previous stage

~~e) repeating b), c), and d) for successive stages until the operand is shifted for processing~~; and

~~g)~~ processing the operand.

2. (original) The method of claim 1, wherein the operands have a bit width of 32 bits.

3. (currently amended) The method of claim 1, wherein the method operation of processing the operand includes,

aligning a least significant bit of the shifted operand to a least significant bit of another operand, the shifted operand having a bit width of one of 8 bits and 16 bits; and

extending the bit width of the shifted operand to 32 bits.

4. (original) The method of claim 3, wherein the method operation of extending the bit width of the shifted operand to 32 bits includes,

padding the extended bit width with a digital value.

5. (original) The method of claim 4, wherein the digital value is selected from the group consisting of 1 and 0.

6. (currently amended) The method of claim 4, wherein the method operation of ~~extending the bit width of the shifted operand to 32 bits~~ padding the extended bit width with a digital value includes,

determining the least significant bit of the shifted operand; and

repeatedly inserting the least significant bit to the shifted operand until the shifted operand has been shifted by the shift value.

7. (currently amended) The method of claim 1 ~~6~~, wherein the ~~four~~ stages include a two to one multiplexer at the first stage, the second stage and a third stage and a four to one multiplexer at a fourth stage.

8. (currently amended) A method for transparently presenting different size operands to be processed, comprising:

providing a first operand having a first bit-width;

determining a bit width of a second operand associated with a processor, the second operand having a greater bit width than the first operand;

transforming the first operand by aligning a least significant bit of the first operand to a lowest bit position of a second ~~transformed~~ operand, the transformed first operand having a bit size equal to the second operand;

padding bits of the transformed first operand;

selecting whether to apply one of a sixteen bit shift value or no shift value to the transformed first operand at a first stage;

shifting the transformed first operand by the selected shift value;

outputting the shifted operand to further stages, wherein a maximum shift value at each successive stage decreases relative to a previous stage; and

processing the operand.

9. (original) The method of claim 8, wherein the second operand is 32 bits wide.

10. (original) The method of claim 8, wherein the first operand has a bit-width selected form the group of 8 bits and 16 bits.

11. (original) The method of claim 8, wherein the method operation of padding bits of the transformed operand includes,

filling higher bits of the transformed operand with a digital value, the higher bits of the transformed operand excluding values associated with the first operand.

12. (currently amended) The method of claim 8, wherein the method operation of padding bits of the transformed first operand ~~transforming the first operand by aligning a least significant bit of the first operand to a lowest bit position of a transformed operand~~ includes,

sign extending the first operand by a value associated with the least significant bit.

13. (original) The method of claim 8, further including:

processing the transformed operand;

transforming the processed operand to a post-processed operand having a first bit-width; and

writing back the post-processed operand to a memory location.

14. (currently amended) A processor, comprising:

an input socket configured to receive and transmit data packets;

a transform unit for manipulating data, the transform unit configured to receive data packets from the input socket;

an arithmetic logic unit (ALU) configured to process multiple operands, the ALU including,

a shifter configured to shift each of the multiple operands in stages, wherein the shifter includes 3-two to one multiplexers and a four to one multiplexer, the 3-two to one multiplexers and the four to one multiplexer defining the four stages; and

an adder configured to add operands;

an alignment block having circuitry configured to align a lowest significant bit of the each of the multiple operands prior to transferring the each of the multiple operands to the ALU, the alignment block circuitry extending the each of the multiple operands to a defined bit width so that a pre-extending bit width of the each of the multiple operands is transparent to the ALU; and

a bus connecting the input socket, the transform unit, the ALU, and the alignment block.

15. (original) The processor of claim 14, wherein the ALU is a 32 bit ALU.

16. (original) The processor of claim 14, wherein the shifter and the adder are supplied with parallel pathways.

17. (original) The processor of claim 14, further including:

a second alignment block downstream from the ALU, the second alignment block configured to return each of the multiple operands received from the ALU to an original bit width.

18. (original) The processor of claim 14, wherein the adder is a 32 bit adder.

Claims 19-20 (cancelled)